



Best Available Copy

IN THE U.S. PATENT AND TRADEMARK OFFICE

Application No. : 10/650,470 Confirmation No.: 6121
Applicants : BIRKETT et al.
Filed : 08/28/2003
TC/A.U. : 2857
Examiner : SUAREZ, FELIX E

Attorney Docket No. : 4981 US

DECLARATION UNDER 37 C.F.R. § 1.131

I, Puneet Suri, do declare and state as follows:

- 1.) That I am properly named as an inventor for the above-identified patent application.
- 2.) That I believe the inventors, including Jon M. Sorenson, Thomas E. Birkett, Puneet Suri, Kai C. Yung, Sylvia H. Fang, Salil Kumar, Benjamin B. Jones, Yuandan Lou, and Cathering E. Frantz, conceived and began reducing to practice in the United States the invention claimed in the above-identified patent application prior to June 4 2002.
- 3.) The attached redacted documents were generated prior to June 4 2002 and include:

Document #1 entitled "Design Document for the GA Analysis Daemon discloses design and implementation details for an Analysis Daemon component used in connection with software components corresponding to the invention disclosed and claimed in the above-identified patent application.

Document #2 entitled "Design Document for Integration of Data Collection and Analysis Applications" discloses design and implementation details for auto-analysis and data collection software functionalities used in connection with components corresponding to the invention disclosed and claimed in the above-identified patent application.

Document #3 entitled "Interoffice Memorandum / Subject GA Auto Analysis Daemon" discloses design and implementation details for an auto analysis daemon and

customizable plugins used in connection with components corresponding to the invention disclosed and claimed in the above-identified patent application.

Document #4 entitled “GA Auto-Analysis Daemon” discloses design and implementation details for an auto analysis daemon used in connection with components corresponding to the invention disclosed and claimed in the above-identified patent application.

Document #5 entitled “GeneMapper 3.0 DAKAR Integration” discloses a protocol manager for performing automated analysis corresponding to the invention disclosed and claimed in the above-identified patent application.

Document #6 entitled “Analysis Group Design Review” discloses details of a data collection component that is used for extracting and analyzing samples corresponding to the invention disclosed and claimed in the above-identified patent application.

Document #7 entitled “SeqScape 2.0 Integration with Foundation Data Collection Automatic Secondary Analysis” discloses details of an automated analysis component corresponding to the invention disclosed and claimed in the above-identified patent application.

5.) The attached information was used in the preparation of a provisional patent application by Applicants' representative, which was filed on August 28, 2002, and designated Application No. 60/407,439 and subsequently converted in a non-provisional application filed on August 28, 2003 and designated Application No. 10/650,470.

Accordingly, based on this information, the subject matter of the above-identified application as claimed was conceived prior to June 4, 2002, and reasonable diligence was

used in constructively reducing the invention to practice, at least by the filing of Provisional Application No. 60/407,439 on Aug 28, 2002.

I hereby declare that all statements made herein of my own knowledge are true, and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Attachment: Redacted Documents #'s 1-7

Ps
Initials









Puneet Suri

2nd Feb 2006

Date

Design Document for the GA Analysis Daemon

1 Revision History

		Tom Birkett	 Consolidated design input from previous documents. Incorporated notes and decisions from meetings with Puneet Suri, Jon Sorenson, Cathy Frantz, Ben Jones and Sylvia Fang.
		Tom Birkett	Updated for comments made at  design/integration meeting. Added picture detailing interactions and communication between analysis daemon components.

2 Abbreviations

AA – Analysis Application
 AD – Analysis Daemon
 ADFL – Analysis Daemon Faceless Component
 ADGUI – Analysis Daemon GUI Component
 ADPI – Analysis Daemon Plugin
 ADFLPI – Analysis Daemon Faceless Plugin
 ADGUIPI – Analysis Daemon GUI Plugin
 GUI – Graphical User Interface
 FDC – Foundation Data Collection for Dakar
 JNDI – Java Naming and Directory Interface
 JMS – Java Message Service
 RMI – Java Remote Method Invocation

3 Introduction

This document describes the design of, and implementation details for, the Analysis Daemon component. This Analysis Daemon component is part of the integration architecture between Foundation Data Collection and downstream analysis applications, specifically GeneMapper 3.0, SeqScape 2.0 and SeqA 5.x. The initial implementation of the Analysis Daemon will assume that all pertinent software components are installed and running on the same physical computer system. This assumption is currently valid based on the designs and deployment plans for all the current applications which comprise the FDC integration architecture.

4 Overview

This section will describe the general steps required for the auto analysis process between FDC and the downstream analysis applications to transpire. It will also briefly touch on implementation details which will be further expanded later in this document.

4.1 FDC Installation

FDC must be the first software component installed on the computer system. As part of its installation, a JNDI server and a JMS server will be installed, and these two components (JNDI and JMS) are required for the auto-analysis components of the downstream analysis applications to be installed and to function correctly.

4.2 Analysis Application Installation

After the installation of FDC is complete, an analysis application which supports the FDC integration architecture will be installed. The installer for the AA will populate the JNDI server previously installed by FDC with application specific information peculiar to that instance of the AA. This application specific information will provide data to FDC that will be used to populate plate records which are specific to that instance of the AA. Each AA will define exactly what data is installed into the JNDI server, and how it will be presented to the user through the plate editor UI of FDC. FDC will provide a generic means of allowing this data to be manipulated by the user within the plate editor UI, and will also provide a more advanced feature whereby the AA's can install their own renderers and editors for the plate editor columns which display their application specific data and make it available for manipulation by the user. This JNDI based data for the AA can be updated either through the GUI of the AA, or through the GUI of the FDC plate editor.

4.2.1 Analysis Daemon Installation

If requested by the user, the installer for the AA, after the successful installation of the AA, will then install the Analysis Daemon. The AD will consist of both a faceless (server) component, and a GUI Application. The faceless (server) component may be implemented as a Windows 2000/NT service, or as a service which starts up after the JNDI and JMS services from FDC have been started. If implemented as a Windows service, it will be implemented with a C++ component and a Java component. Otherwise it will be a Java based daemon which may be managed by the FDC Service Console Application which currently manages the JNDI and JMS services. In any case, it will comprise the Analysis Daemon Faceless Component. This ADFL will provide generic functionality which will be augmented and customized by the Analysis Daemon Faceless plugin for a particular AA. The Analysis Daemon Faceless plugins will be implemented in Java. The GUI Application of the AD will be implemented as a Java application. Likewise, the ADGUI will provide generic functionality which will be extended by the Analysis Daemon GUI plugins. These ADGUIs will be implemented in Java and will present an AA specific UI panel to the user.

The installer for the AA will install the Analysis Daemon Faceless component and the Analysis Daemon GUI component if they are not already installed. It will then install the application specific plugins for the Analysis Daemon. If implemented as a Windows service, then the analysis daemon faceless component will be configured to auto-start at boot time. After the installation of the Analysis Daemon is complete, the user will be requested (forced) to reboot their computer system.

4.2.2 Analysis Daemon Startup

The Analysis Daemon will start up and will immediately attempt to connect to the installed JNDI server and the installed JMS server, and will continue this pursuit until successful. Upon successful startup of the Analysis Daemon, all installed Analysis Daemon Faceless plugins will be loaded and initialized. This will cause the JNDI server to be queried by the AD for the FDC instrument name, and for each installed plugin a subscription to the JMS server will be established based on a topic whose name is constructed from the instrument name and the AA instance name. The Analysis Daemon will also create topics in the JMS server, if not already present, that will be used to support communication between the GUI component of the AD and the faceless component of the AD. There will be topics created for generic communication between the ADFL and the ADGUI, as well as topics created for each ADFLPI and ADGUIPI. This feature will provide for direct communication between an Analysis Application GUI plugin



and its corresponding faceless plugin. Hence, UI manipulations made within the context of the AA GUI plugin can be directly “passed” to the faceless plugin for that AA, and status from the faceless plugin can be directly accessed by the GUI plugin. The plugins will call a well defined interface provided by the AD to facilitate this functionality, and the low level publishing and subscribing to the JMS server will be handled by the generic implementation of the AD.

4.3 Completion of FDC Run / Analysis Group

At some point after the successful installation of both FDC and an AA configured to support auto-analysis, an FDC run will be completed or an Analysis Group will be completed. This completed run/analysis group will have been configured (from within FDC) to publish a corresponding Message to the appropriate topic on the JMS server which runs on this same computer. If the AD is running, or upon successful startup of the AD, the AD will have subscribed to this published Message for this specific topic and will receive it via its installed message listener. If the type of Message (run completed or analysis group completed) that has been received by the AD has been “requested” by the appropriate AA plugin, then the AD will add this Message to its persistent message queue, and then notify the AA plugin that a new “batch” job is available. There will be a mechanism implemented whereby an AA plugin, even though it is not currently busy, can refuse to accept an available “batch” job and can cause the AD to reschedule this batch job at a later time.

4.4 Analysis Daemon to Analysis Application Communication

When the ADFLPI accepts a batch job for processing, it should update the state of this job to “in process” (or the equivalent). The ADGUI, if running, will display the status of all jobs in its persistent message queue, but managing the status of a particular job will be the responsibility of the ADFLPI. Once a job is accepted for processing by the ADFLPI, it becomes the responsibility of the ADFLPI plugin to determine exactly what operations transpire to process this job successfully. These responsibilities include maintaining the state of the batch job, determining how the ADFLPI and AA will interact to process the batch job, and determining how the samples which comprise the batch job should be “handled”. The ADFL/ADGUI will perform no specific sample processing; they will simply maintain the persistent message (job) queue, handle the interactions with the JMS server, and provide rudimentary message queue display and management facilities (more on this will follow). The communication protocol between the AA and the ADFLPI will be up to the author(s) of these software components to decide.

While a “batch” job is being processed by the ADFLPI, it is the plugin’s duty to maintain whatever state information is pertinent to that job. This state information should be communicated back to the ADFL for updating the persistent job list. The ADGUI, if active, will be notified that its data model (persistent job list) has been updated and it should display the new status of this particular job.

4.5 Analysis Daemon GUI Display

The ADGUI will be implemented as a Java application which the user can run at any time. It will consist of a main, general panel which will display the current status of all “batch” jobs currently resident in the AD persistent job queue for this computer. Status display will include “job” name, AA instance name of job “owner”, current processing state, # of samples in job and the date when the job was received by the AD. From this panel the user will be able to (request to) delete jobs whose current processing status is “Completed”. Any additional functionality provided on the



jobs in the queue will be the responsibility of the ADGUIPI for a particular AA. The ADGUIPI will present a separate display panel to the user within the same graphical window presented by the ADGUI (a tabbed panel display will be the most likely implementation). The ADGUIPI will be provided with a list of jobs that are targeted to that AA plugin, and the display of these jobs will be entirely dependent on the ADGUIPI. The AD will provide interfaces for allowing the jobs (to be requested) to be rescheduled, resubmitted or removed within the context of the persistent job queue maintained by the AD. However, the UI manipulations required to effect these operations will be the responsibility of the ADGUIPI to provide. As mentioned earlier, the AD will provide interfaces to the ADGUIPI and ADFLPI so that each may "post" events (messages) to the other. This will allow the author of these plugins to implement direct communication between the GUI and Faceless components; a UI action within an ADGUIPI could cause a currently processing job to be cancelled by the ADFLPI.

4.6 Analysis Daemon Message Queue Management

It is the responsibility of the AD to maintain a persistent message queue of the messages which it receives from the JMS server and which have been targeted at loaded AA plugins (A message is targeted to an AA plugin if the AA plugin has registered with the AD for messages of this type. Currently there are but two types of messages defined, run completed or analysis group completed.) This persistent message queue must survive AD shutdown and restart, system restart and any catastrophic events such as system crashes. This persistent message (job) queue can be thought of as the data model of the Analysis Daemon.

5 "Generic" Analysis Daemon Implementation

5.1 Faceless Windows 2000/NT Service or Java Daemon

The Analysis Daemon Faceless component may be implemented as a Windows 2000/NT service. (Is this a requirement? The Analysis Daemon requires both the JNDI server and the JMS server, so should it not simply be a Java based service which starts up after these previous two services? FDC already provides a Service Console Application for managing the services required by FDC. This would allow the user to manage all of these software components from one place. After speaking with the developer of the Service Console Application for FDC, it would be trivial to have the ADFL work within the context of the FDC Service Console Application.)

5.1.1 C++ Component of Service

If implemented as a Windows service, then the C++ component will handle the "native" requirements of a Windows service. It will handle access to the Window registry and provide the service-specific functions that must be provided in the native code if it is to be installed as a Windows service. It will also provide all of the functions that are required for creating, running and removing a Windows service, invoking the JVM and inserting and retrieving registry data. The Java component of the AD will be started up by this C++ code.

5.1.2 Java Component of Service

The Java component of the ADFL will provide the true functionality of the analysis daemon faceless component. It will query the JNDI service for configuration information (at least instrument name), will register message listeners and create topics if required with the JMS



service for messages published by FDC, ADFLPIs and ADGUIPIs, and will maintain a persistent message queue of the messages which it receives from FDC (the queuing mechanism and persistent data store for these message is TBD).

A set of APIs will be provided which the ADFLPIs will use to access the resources which are maintained by the ADFL. These APIs will be provided in the Java interface "IAnalysisDaemonService", an initial version of which appears at the end of this document. This interface will encapsulate all that is required of the ADFL by the ADFLPI. Additionally, each ADFLPI will be required to implement the interface "IAnalysisDaemonServicePlugin". This interface, also appearing at the end of this document in an initial version, will define the APIs through which the ADFL interacts with the ADFLPI.

5.2 GUI Display

As previously stated the ADGUI will be implemented as a Java application. Upon startup it will construct its main window, install its General panel and then query the ADFL (via the JMS service) for the current persistent message queue which the ADFL maintains. When the queue of Messages (jobs) is returned, it will populate its general panel with this list. If the JMS service or the ADFL are not running when the ADGUI application is executed, after some period of time it will warn the user that something is amiss, and request that the user first check that the required services (JNDI and JMS services) are running and available (the FDC Service Console Application could be used for this purpose).

Once the application has successfully started up, it will load its installed ADGUIPIs and initialize them. As part of their initialization, each ADGUIPI will be supplied with the list of "jobs" peculiar to that plugin that are currently known to the ADGUI. As new "jobs" arrive at the ADFL (or are deleted, rescheduled or resubmitted), it will notify the ADGUI (through the publishing of a JMS message) that its persistent job list has been updated, and eventually the ADGUIPI responsible for them will be notified of their existence (change in state).

A set of API's will be provided which the ADGUIPIs will use to access the resources that it requires of the ADGUI/ADFL. These APIs will be provided in the Java interface "IAnalysisDaemonGUI", an initial version of which appears at the end of this document. This interface will encapsulate all that is required by the ADGUIPI to interact with the ADGUI/ADFL. Additionally, each ADGUIPI will be required to implement the interface "IAnalysisDaemonGUIPlugin". This interface, also appearing in an initial version at the end of this document, will define the APIs through which the ADGUI interacts with the ADGUIPI.

6 Application Specific Analysis Daemon Plugin Implementation

AD plugins will be written in Java by developers who have reason to be informed of the run completed or analysis group completed messages generated by FDC. These developers may also need to have in-depth (uncanny) knowledge of how a particular AA behaves in auto-analysis mode! Generally a plugin implementation to the AD will be comprised of two software components, and ADFLPI and ADGUIPI. However, the GUI component may be considered extraneous and not provided by the implementation.



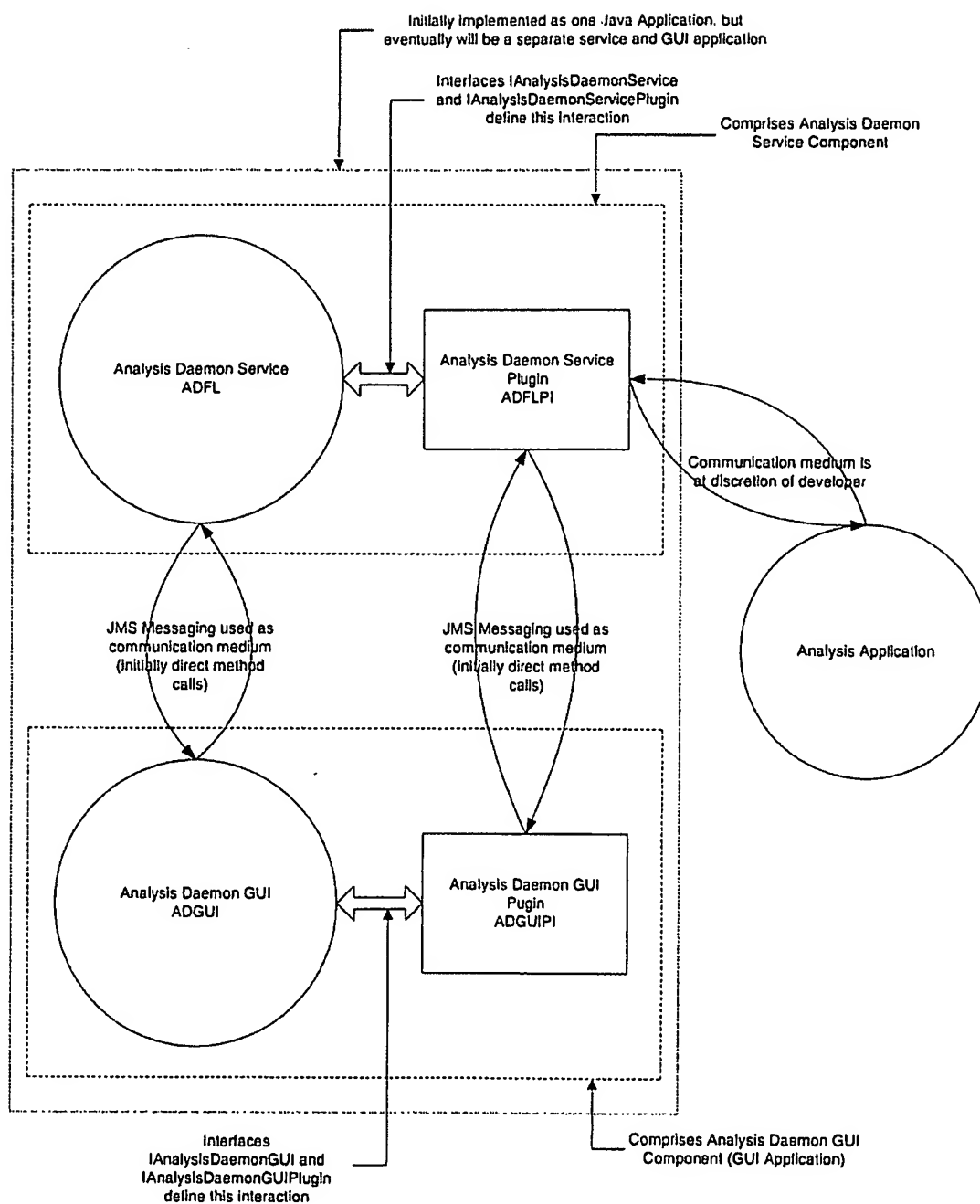
6.1 Faceless Plugins

Each ADFLPI will implement its required interface, `IAAnalysisDaemonServicePlugin`, and will interact with the ADFL through an interface, `IAAnalysisDaemonService`. An ADFLPI will handle all interactions with the AA for which it is designed. While it is busy processing a “job”, it will maintain the current job status, and insure that this status is faithfully propagated back to the ADFL so that the status of this job can be made available to the GUI components of the system (it may suffice to simply update the state to “processing” until the job completes). It may interact with its corresponding GUI component, but is not required to do so. Most likely, when there is no current job for it to process, it will remain harmlessly inactive!

6.2 GUI Plugins

Each ADGUIPI will implement its required interface, `IAAnalysisDaemonGUIPlugin`, and will interact with the ADGUI through the interface “`IAAnalysisDaemonGUI`”. The GUI plugin will be “initialized” with a reference to its `IAAnalysisDaemonGUI` along with the current list of jobs that have been targeted to it. Once the ADGUIPI is initialized, the ADGUI will request a reference to the UI component through which the ADGUIPI will be displaying the job list. As the job list is “updated” (either through additions, deletions or rescheduling/reordering) the GUI plugin will be notified of these events. It may also trigger these events through its UI. Whenever the state of the persistent job list changes, the ADGUI will be notified of this event and propagate this change to the ADGUIPI with an updated list of jobs for that plugin.

7 Graphical Representation of Analysis Daemon Interactions



8 Published Java Interfaces

The following are initial versions, and will change often during development. They are currently here to give some indication to potential plugin authors of what may be expected.



8.1 IAnalysisDaemonService

This is the interface implemented within the ADFL. An object which implements this interface will be passed to an ADFLPI via its "initialize" method. The ADFLPI should maintain a reference to this object.

```
public interface IAnalysisDaemonService
{
}
}
```

8.2 IAnalysisDaemonServicePlugin

```
public interface IAnalysisDaemonServicePlugin
{
}
}
```

8.3 IAnalysisDaemonGUI

```
public interface IAnalysisDaemonServicePlugin
{
}
}
```

8.4 IAnalysisDaemonGUIPlugin

```
public interface IAnalysisDaemonServicePlugin
{
}
}
```

DESIGN DOCUMENT	
PRODUCT NAME AND PRODUCT VERSION	
Saved As: DC_Automation_Analysis.doc	
CURRENT REVISION: [REDACTED]	LAST EDIT: [REDACTED]
PRINTED: [REDACTED]	PAGE 1 OF 7

DESIGN DOCUMENT FOR INTEGRATION OF DATA COLLECTION AND ANALYSIS APPLICATIONS

REVISION HISTORY			
Revision	Date	Author	Description of Change Implemented
[REDACTED]	[REDACTED]	Sylvia Fang	Initial Draft
[REDACTED]	[REDACTED]	SYLVIA FANG	ADDED MORE DESCRIPTION FOR ANALYSIS GROUP

DESIGN DOCUMENT	
PRODUCT NAME AND PRODUCT VERSION	
Saved As: DC_Automation_Analysis.doc	
CURRENT REVISION: [REDACTED]	LAST EDIT: [REDACTED]
PRINTED: [REDACTED]	PAGE 2 OF 7

TABLE OF CONTENTS

1.0 REVISION HISTORY DETAIL.....	3
2.0 INTRODUCTION.....	3
3.0 DESIGN	3
3.1 Shared Data Repository (JNDI Server).....	4
3.2 Dynamic Invocation of Analysis Application Domain Component (Protocol Editor)	6
3.3 Auto-Analysis through Event Notification.....	6
3.3.1 Analysis Group.....	6
3.4 Installation Consideration	7
3.5 AB Components Dependencies	7
4.0 OPEN ISSUES.....	7
5.0 REFERENCES.....	7

DESIGN DOCUMENT	
PRODUCT NAME AND PRODUCT VERSION	
Saved As: DC_Automation_Analysis.doc	
CURRENT REVISION	LAST EDIT
PRINTED	PAGE 3 OF 7

1.0 REVISION HISTORY DETAIL

Revision	Description of Changes Implemented
1.0	Initial Draft
2.0	Added more description for analysis group based on the discussion on [REDACTED]

2.0 INTRODUCTION

Historically when there was a new type of analysis application being released, data collection software would also need to be rev to support the new analysis application. Data collection software hard-coded the information that needed to be gathered for each analysis application it supported. Eventually it becomes cumbersome and time-consuming, because releasing an analysis application also means another revision of data collection software. To alleviate the coupled release schedules between data collection software and analysis applications, we want to design for a plug-in type of integration such that data collection software dynamically gathers all information from the analysis applications based on a pre-agreed upon contract. As long as the analysis applications follow the contract, data collection software conceivably will be able to support all future release of analysis applications without any modification of source codes.

Another important goal of the integration is to be able to perform auto-analysis. Historically, sample file is the only data transfer vehicle between data collection and the analysis applications. Users need to manually import the sample files generated from data collection into the analysis application, specify the analysis parameters, and perform secondary analysis. Part of the integration effort is for data collection to gather all information the analysis application needs to perform secondary analysis, and through event notification auto-analysis is kicked off without having users to launch the analysis application user interface.

3.0 DESIGN

The overall design of the integration is best depicted in the following sequence diagram.

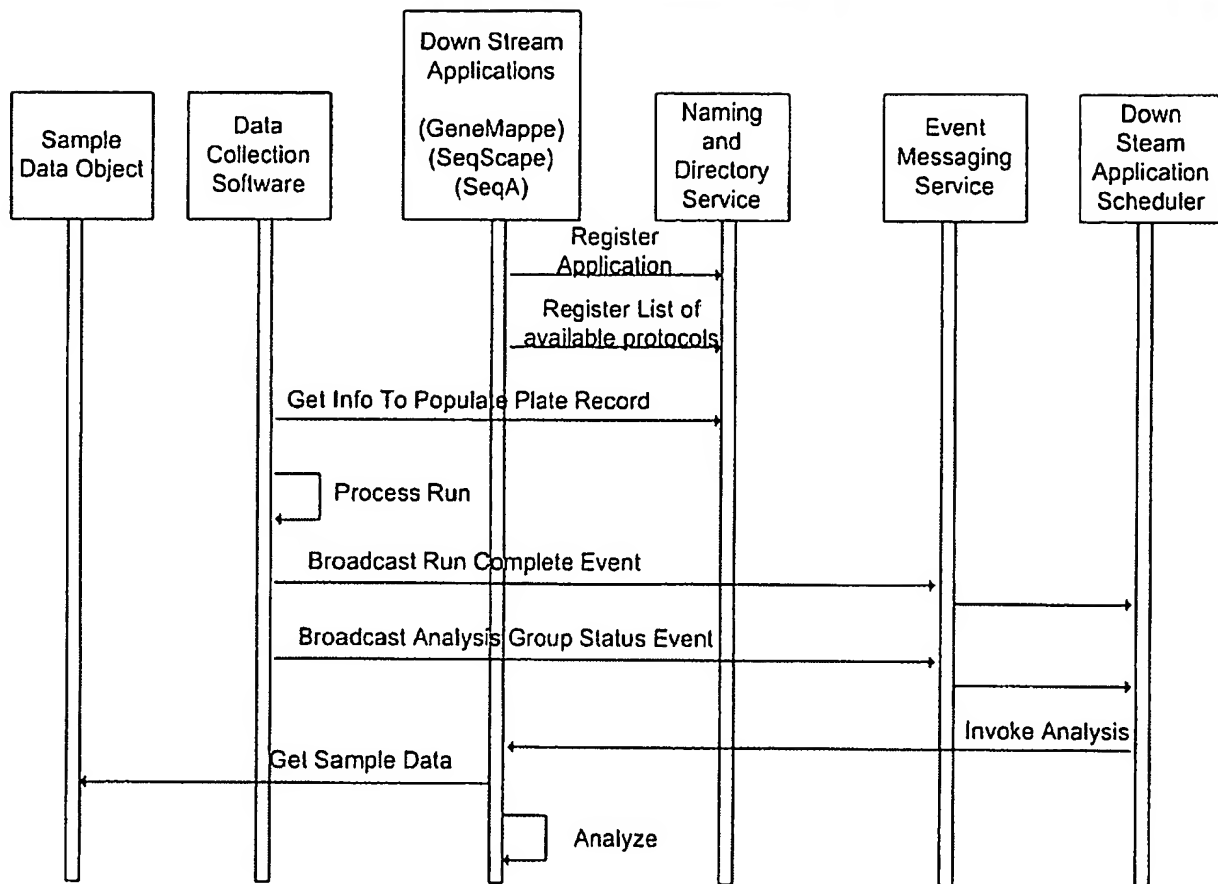


Figure 1 Auto-analysis sequence diagram

The analysis application registers itself and shared data (i.e. analysis protocols, sample infos..) to JNDI server. Foundation Data Collection (FDC) software at run time constructs the plate record (sample sheet) specific to an application type with the sample information from JNDI server. At the end of the run, and at the end of the completion of an analysis group, message is broadcast from FDC. Each application has a message listener service running, ready to receive the messages, and kicks off analysis when message is received.

3.1 Shared Data Repository (JNDI Server)

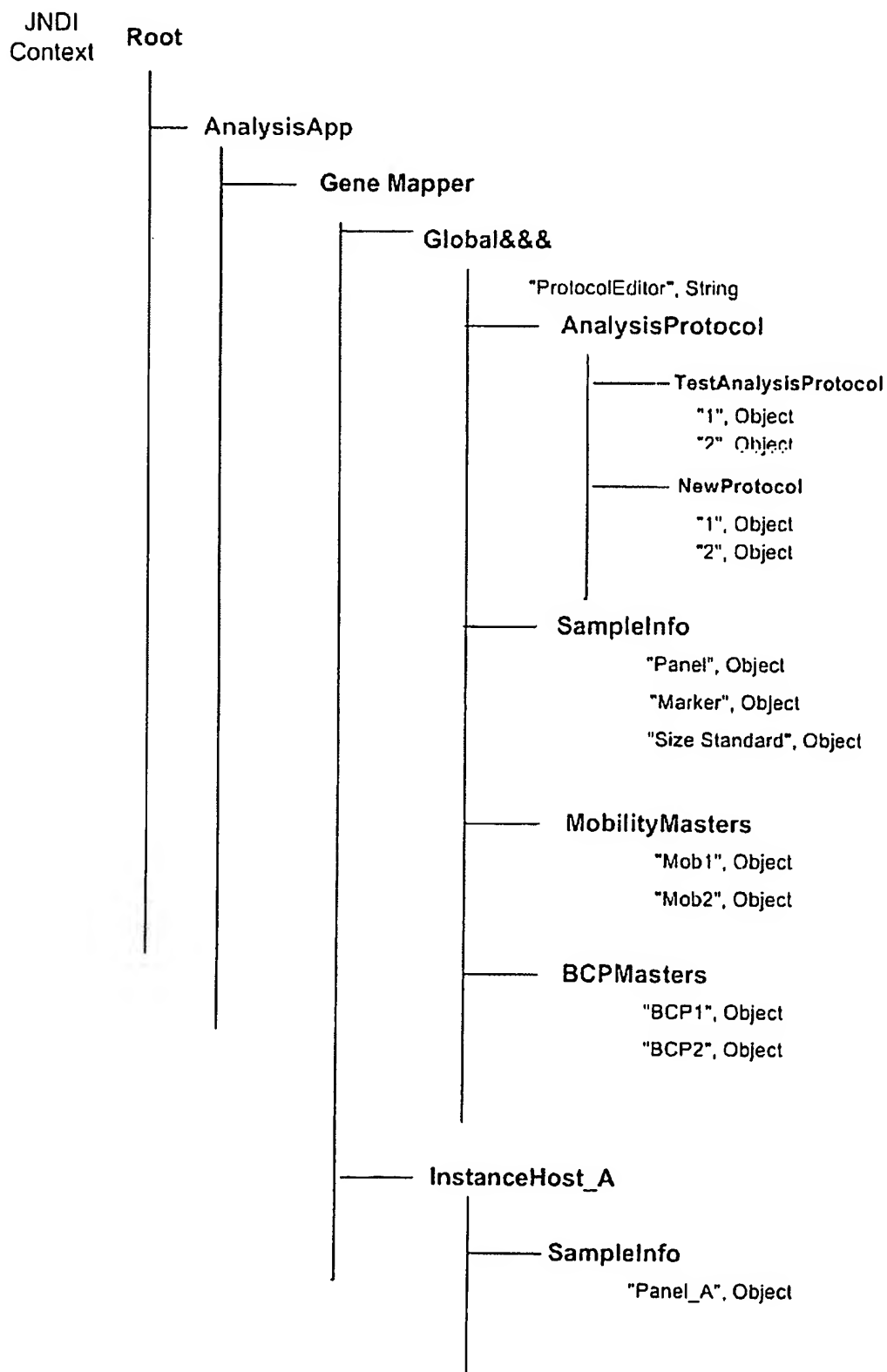
When data sharing is required, JNDI Server (a naming and directory server) serves a convenient mean as data repository. Persistent JNDI server is part of Foundation Data Collection (FDC) installation. It can be accessed by remote clients, and it provides shared data storage without database requirement.

JNDI naming service consists of APIs to look up, binding/unbinding objects, creating and removing contexts. The JNDI storage hierarchy is very similar to hard drive directory structure with contexts representing folders, and bound objects representing files.

The proposed JNDI hierarchy is depicted in the following diagram.

DESIGN DOCUMENT	
PRODUCT NAME AND PRODUCT VERSION	
Saved As: DC_Automation_Analysis.doc	
CURRENT REVISION: [REDACTED]	LAST EDIT: [REDACTED]
PRINTED: [REDACTED]	PAGE 5 OF 7

Proposed JNDI Context Hierarchy



DESIGN DOCUMENT	
PRODUCT NAME AND PRODUCT VERSION	
Saved As: DC_Automation_Analysis.doc	
CURRENT REVISION: [REDACTED]	LAST EDIT: [REDACTED]
PRINTED: [REDACTED]	PAGE 6 OF 7

Figure 2 JNDI hierarchy

The text in blue means it's a fixed name based on the agreement between FDC and analysis applications. The text in black is populated at run time and the content is presented as examples. At the root of JNDI tree, an "AnalysisApp" context is created, and under it each sub-context represents a type of registered applications. Since some applications (for example, GeneMapper) have instance-specific (each installation is an instance) information to be shared with FDC, each analysis application has sub-context "Global&&" to store global (instance-independent) data, as well as instance context represented as the installation machine name, to store instance specific data.

3.2 Dynamic Invocation of Analysis Application Domain Component (Protocol Editor)

Users are allowed to create new, edit an existing, import / export, and delete analysis protocols from within FDC software, in addition to using the analysis applications. Each analysis application has its own definition of analysis protocol. The design goal is: whenever there's change in the analysis protocol definition, FDC will be updated without requiring any code changes.

Analysis application implements its own domain specific analysis protocol editor. The fully qualified class name of the protocol editor is registered into JNDI. FDC at run time will be able to invoke the protocol editor by its class name with Java Reflection APIs as long as the protocol editor jar file is in the FDC class path. During the installation of the analysis application, the installer will read from a configuration file from a pre-determined location the FDC installed directory and add the jar file to FDC class path. This is an installation issue and is discussed in [Section 3.4](#).

All analysis protocol editors need to conform to the following constructor signatures. The first constructor is to invoke the editor to add new protocols; the second constructor is to load the parameters for ProtocolName for edit purpose. The argument AppType is the name of the analysis application registered in the JNDI tree.

```
public ProtocolEditor(String JNDIServerURL, String AppType);
```

```
public ProtocolEditor(String JNDIServerURL, String ProtocolName, String AppType);
```

All analysis protocol editors will keep track of their versions internally, independent of FDC.

3.3 Auto-Analysis through Event Notification

Secondary analysis is invoked through message notification. FDC will publish events that are of interest to analysis applications. The application will determine whether analysis will be performed based on the event content.

3.3.1 Analysis Group

Secondary analysis involves a group of samples. This grouping of the samples for auto-analysis is called "Analysis Group". User defines and creates analysis groups in application instance, and the application populates the names of analysis groups into the JNDI, and updates the entries whenever user creates new ones. During the installation of each application instance, a unique JMS Topic specific to each instance will be created, and a daemon listening to the topic will be instantiated.

FDC gets the names of analysis groups from JNDI, and provides GUI for users to choose the destination for the sample files, and to indicate whether all samples in the analysis groups are completed. When FDC processes all runs in the analysis group, FDC gets the instance topic name

from the JNDI and publishes event to the topic. The application instance the analysis group belongs to receives and interprets the messages, and invokes auto-analysis.

User will be allowed to create new analysis groups in FDC. However, because the analysis group created in FDC is not tied to any application instance, FDC will not notify any application instances.

The published message is a message object, with the following class definition:

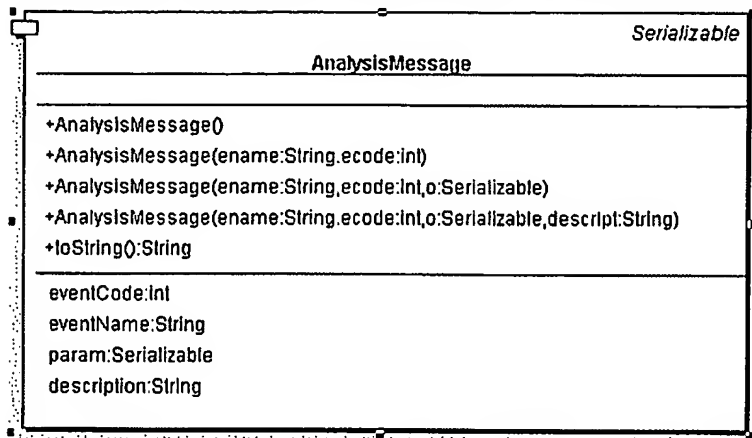


Figure 3 the Definition of the message object to be published in the JMS topic

The events that will be published are: RunCompleted, AnalysisGroupCompleted. The content of the message is TBD.

3.4 Installation Consideration

As part of integration process, applications need to be able to share information among them. In order to share installation configuration information, all applications will write to a property text file, located at TBD (per AB installation guideline).

3.5 AB Components Dependencies

FDC is architected based on AB Components. Due to the tight release schedules of the analysis applications, the first release of integration effort will be JMS (instead of AB Request/Reply) based.

4.0 OPEN ISSUES

5.0 REFERENCES

Minutes, Automation/Integration Issues among Data Collection and Downstream applications. 3/26/2001.
Design Document for SeqA 5.0 (JSA) and Foundation Integration

INTEROFFICE MEMORANDUM

TO:

[REDACTED]

FROM: PUNEET SURI

SUBJECT: GA AUTO ANALYSIS DAEMON

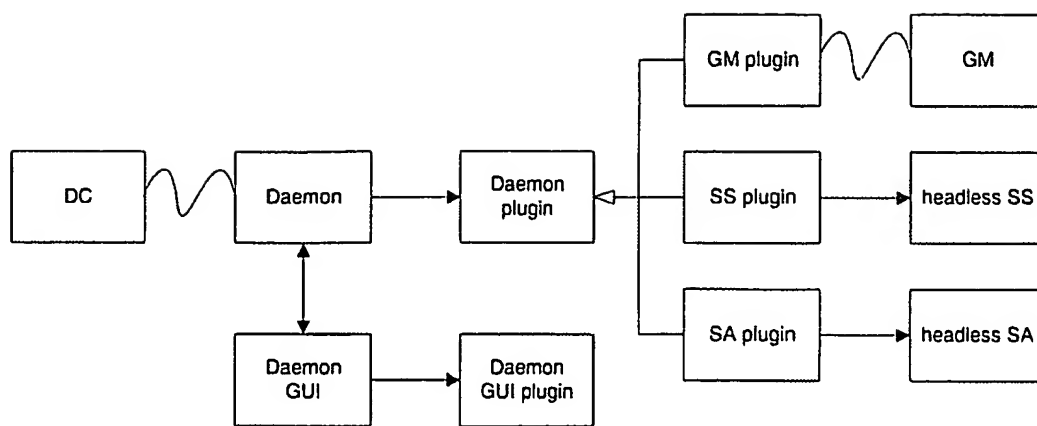
DATE:

[REDACTED]

CC:

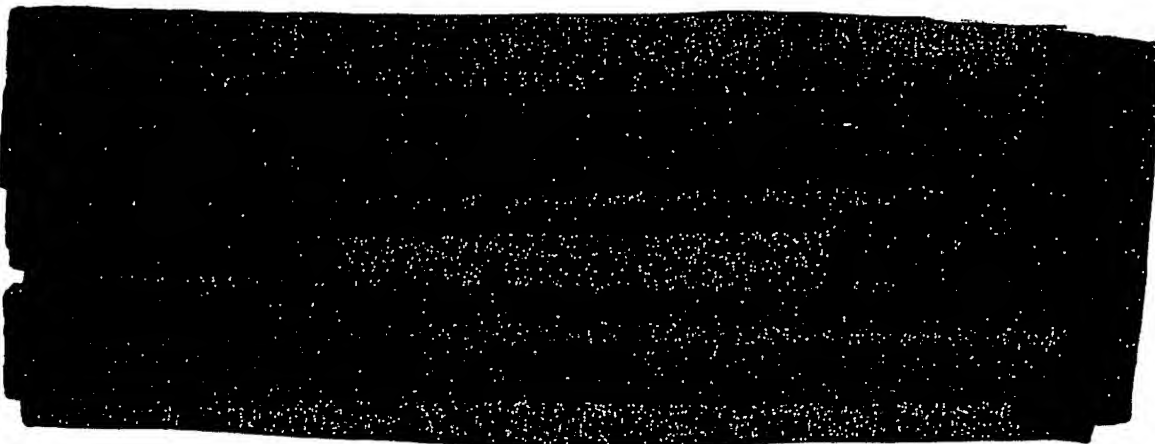
[REDACTED]

[REDACTED] the following diagram on the whiteboard summarizing the role of the daemon and customizable plugin components:



(curly lines represent messaging.)

Current JMS messages anticipated to be processed by the daemon are "Analysis Group done" and "Run done" messages. For both GM and SS, user name and password will be contained in the analysis group (encrypted in the JMS message). Each sample will be associated with an analysis group and the contents of the analysis groups for a run will be passed through JMS so that SeqScape will be able to extract different user names/passwords for different samples in the same run. GM only needs the "analysis group done" message and the name and contents of the analysis group to proceed with auto-analysis.



GA Auto-Analysis Daemon

[REDACTED]

The daemon's job is to listen for messages from DC and schedule processing with the appropriate analysis application instance. The functionality that the daemon provides that is common to all GA applications is the handling of batches of sample files to be auto-analyzed.

[REDACTED]

Each application instance registers itself in JNDI. This provides a unique JMS Topic for each application instance used for automation messaging.

The daemon is configured to listen for messages for the JMS Topic associated with the application instance(s) it is supporting.

The daemon is actually two pieces of related software, an NT/Win2000 service that performs the messaging, queuing, and other batch management, and a separate GUI application that communicates with the service to display the message/batch queue and results of processing (more on this below). One daemon will support all of the GA application instances on a machine. Since it is running as a service, it will be started automatically when the machine is booted, and will stop when the machine is shut down. It can also be controlled manually (started, stopped, etc) in the Windows control panel for services.

[REDACTED] Using a Win service is OK for GM, although GM workflow does not require it. The messages for the daemon will come from a JMS server that is not a service

[REDACTED]

[REDACTED]

The daemon can queue messages and track the status of their processing. Each message is considered a batch job, whether it contains a single sample, samples from an analysis group, or an entire run of samples.

[REDACTED]


The daemon has to be able to communicate with the application instance in some fashion to start the batch processing and to get results back. Starting the batch processing is fairly application specific, and will therefore be handled via a plug-in provided by each application development team. For example in processing SeqScape batches, the daemon plug-in should move the sample files to the SeqScape instance's staging area and start a

new instance of SeqScape in headless automation mode. Among other things, the GeneMapper daemon plug-in would pass the batch to an already running instance of GeneMapper, or start GeneMapper if one is not already running.

The daemon listens for two types of results messages from the application instances – results of processing the entire batch, and results for each sample within the batch. The daemon will track and store these results. The related GUI application can communicate with the daemon to get these results for display purposes. Details about the analysis application instance's processing of the batch should be reported/displayed in the application itself.

The analysis application should send the batch results message back to the daemon when processing is complete, and it should contain the batch Id and a preformatted string of the results (such as "Processing Complete With Errors", "Error – Password Not Valid", etc.).

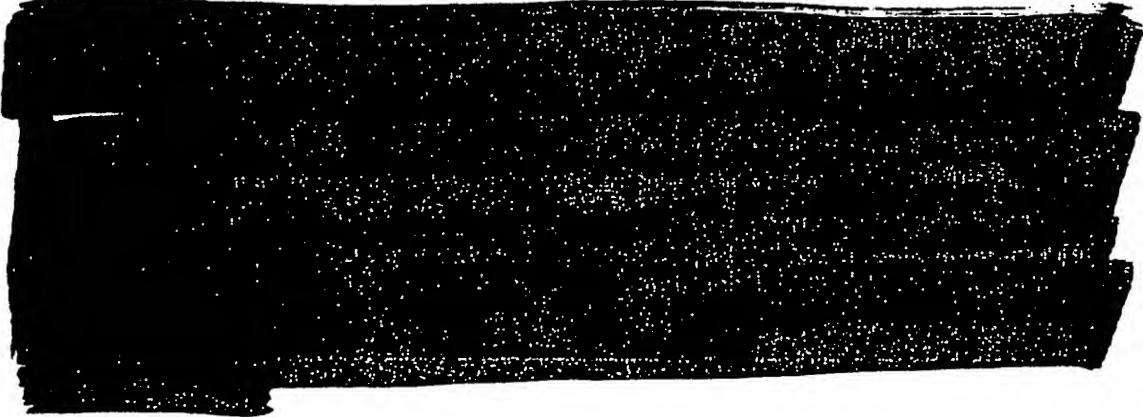

The analysis application should send sample results back to the daemon after the sample has been analyzed, and should contain the batch Id, a boolean indicating success/failure, and an optional preformatted results/error string.



The GUI application

The GUI application will be very similar to that described in the paper "GeneMapper 3 Integration with Foundation Data Collection: Automatic Analysis for Dakar"

It will show a table with columns for the batch name, user name, number of samples in the batch, date/time in, and the current status for the batch. Buttons under the table will allow the user to resubmit the selected batch, delete the selected batch, and see detailed status for the samples in the selected batch. Should we allow canceling a running batch process?



I suggest that we provide for possible future complete configurability of the GUI, that is, an app installation would be able to have an app-specific GUI for the daemon, at least as a GUI plug-in to the GUI that handles all apps. Perhaps the all-app GUI could have separate tab panes for each app instance or app type. Because the GUI and the app-specific daemon plug-in can get into app-specific job control and workflow areas, we can't expect that one GUI will serve for all apps forever. I see below that GUI plug-ins will be possible.

Since the daemon tracks the success/failure of each sample within a batch, it is possible for it to either resubmit the entire batch or only resubmit the failed samples of the selected batch. Should this be a configurable option?

If samples get into a GM project, then re-submission of the batch job is not necessary. If samples somehow fail to get imported into a GM project, then re-submission might be necessary. These cases can be handled by messages at the job level, so the (GM) daemon doesn't need to track each sample.

When the GUI application is running, the daemon will pass it the necessary status messages for keeping the display up to date with the on-going application results messages and new batch messages from DC.

The batch details panel/dialog of the GUI application will display information about the batch and a table of the results for each of the samples. The table will have columns for the sample name, Boolean for success/failure, and the pre-formatted results string received from the application.

It would also be possible for the analysis applications to provide plug-ins for the GUI application to control the display of results or application specific interaction with the daemon, but this should not be necessary. The daemon will be easier to maintain and easier for customers to understand/use if it is kept simple and generic across all GA applications.

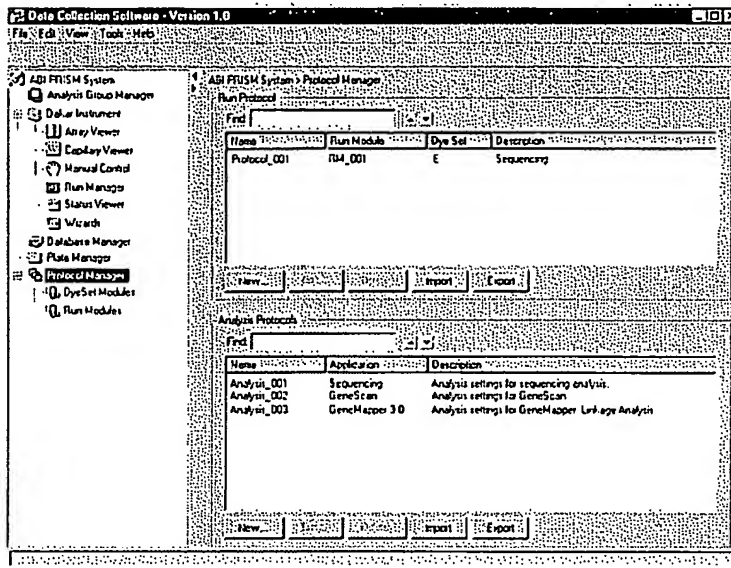
Summary

The idea is to have one daemon that can support all of the common requirements of the different GA applications. This common functionality includes receiving automation

(batch) messages from DC, queuing the messages, making sure the analysis applications are appropriately notified of the batch messages, and displaying basic results on a batch and sample file basis. The daemon will follow industry standards and run as a Windows service, providing always-on functionality, and a separate GUI utility application will be used for monitoring and interacting with the daemon service.

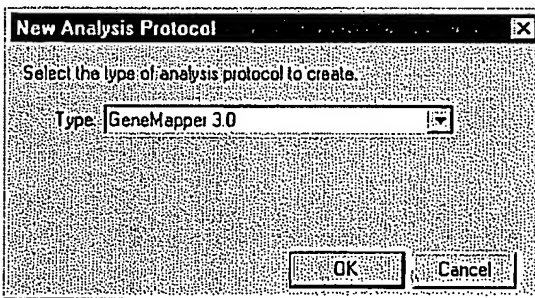
8. GeneMapper 3.0 DAKAR Integration

8.1 Analysis Protocols Dakar shall be able to access GeneMapper 3.0 Analysis Protocols through the Dakar Protocol Manager



Clicking New or Open (for a GeneMapper protocol) would open the Analysis Protocol Editor.

If New is clicked, then:



1. Choose the GeneMapper 3.0 Protocol.

New Analysis Settings [X]

Select the analysis type:

☐ HID ☐ SNaPshot

☒ Microsatellite (Linkage Mapping) ☐ Cystic Fibrosis

OK Cancel

2. Select the analysis type.

GeneMapper Analysis Protocol Editor - Microsatellite [X]

General | Threshold | Allele | Peak Detector | Peak Quality | Quality Flags

Analysis Protocol Description

Name:


Description:

Instrument:

Analysis Type: Microsatellite

OK Cancel

3. Display the GeneMapper Analysis Protocol Editor.



8.2 Plate Document

Dakar shall be able to create GeneMapper-compatible Plate Documents through the Dakar Plate Manager

Plate Editor

File Edit

Plate Name: GeneMapper_001 Operator: _____

Plate ID: 100 Owner: _____

Well	Sample Name	Comment	Sample Type	Analysis Group	Standard Dye	Panel	Size Standard
A1	Sample_001		Sample	Project_001	Red	LMS_Panel01	GS500
D1	Sample_002		Control	Project_001	Red	LMS_Panel01	GS500
C1	Sample_003		Ladder	Project_001	Red	LMS_Panel01	GS500
D1	Sample_004		Sample	Project_001	Red	LMS_Panel01	GS500
E1	Sample_005		Sample	Project_001	Red	LMS_Panel01	GS500
F1	Sample_006		Sample	Project_001	Red	LMS_Panel01	GS500
G1	Sample_007		Sample	Project_001	Red	LMS_Panel01	GS500
H1	Sample_008		Sample	Project_001	Red	LMS_Panel01	GS500
A2	Sample_009		Sample	Project_001	Red	LMS_Panel01	GS500
B2	Sample_010		Sample	Project_001	Red	LMS_Panel01	GS500
C2	Sample_011		Sample	Project_001	Red	LMS_Panel01	GS500
D2	Sample_012		Sample	Project_001	Red	LMS_Panel01	GS500
E2	Sample_013		Sample	Project_001	Red	LMS_Panel01	GS500
F2	Sample_014		Sample	Project_001	Red	LMS_Panel01	GS500
G2	Sample_015		Sample	Project_001	Red	LMS_Panel01	GS500
H2	Sample_016		Sample	Project_001	Red	LMS_Panel01	GS500
A3	Sample_017		Sample	Project_001	Red	LMS_Panel01	GS500
B3	Sample_018		Sample	Project_001	Red	LMS_Panel01	GS500

Description: _____

AutoAnalysis

☐ Automatically analyze the samples on this plate.

GeneMapper 3.0 Installation: _____

Username: _____

Password: _____

Plate Editor

File Edit

Plate Name: GeneMapper_001 Operator: _____

Plate ID: 100 Owner: _____

Well	Analysis Group	Standard Dye	Panel	Size Standard	Run Protocol	Analysis Protocol
A1	Project_001	Red	LMS_Panel01	GS500	RP_001	AP_001
D1	Project_001	Red	LMS_Panel01	GS500	RP_001	AP_001
C1	Project_001	Red	LMS_Panel01	GS500	RP_001	AP_001
D1	Project_001	Red	LMS_Panel01	GS500	RP_001	AP_001
E1	Project_001	Red	LMS_Panel01	GS500	RP_001	AP_001
F1	Project_001	Red	LMS_Panel01	GS500	RP_001	AP_001
G1	Project_001	Red	LMS_Panel01	GS500	RP_001	AP_001
H1	Project_001	Red	LMS_Panel01	GS500	RP_001	AP_001
A2	Project_001	Red	LMS_Panel01	GS500	RP_001	AP_001
B2	Project_001	Red	LMS_Panel01	GS500	RP_001	AP_001
C2	Project_001	Red	LMS_Panel01	GS500	RP_001	AP_001
D2	Project_001	Red	LMS_Panel01	GS500	RP_001	AP_001
E2	Project_001	Red	LMS_Panel01	GS500	RP_001	AP_001
F2	Project_001	Red	LMS_Panel01	GS500	RP_001	AP_001
G2	Project_001	Red	LMS_Panel01	GS500	RP_001	AP_001
H2	Project_001	Red	LMS_Panel01	GS500	RP_001	AP_001
A3	Project_001	Red	LMS_Panel01	GS500	RP_001	AP_001
B3	Project_001	Red	LMS_Panel01	GS500	RP_001	AP_001

Description: _____

AutoAnalysis

☐ Automatically analyze the samples on this plate.

GeneMapper 3.0 Installation: _____

Username: _____

Password: _____

Column	Description
Well	Read only Well Position
Sample Name	Editable, alphanumeric. Name of the sample
Comment	Editable, alphanumeric. Comment for the sample.
Sample Type	Combo box. List contains the various sample types compatible with GeneMapper. List includes Sample, Control, Ladder. GeneMapper must register these objects in JNDI.
Analysis Group	Dakar-specific, combo box. Logic for auto analysis: <ul style="list-style-type: none"> If GeneMapper does not contain a project with the same name as a sample's Analysis Group, then GeneMapper shall automatically create a project and name it with the Analysis Group's name and add the sample to that project. If GeneMapper does contain a project that has the same name as a sample's Analysis Group, then GeneMapper shall add the sample to the existing project.
Standard Dye	Combo box. List contains the dye colors supported by GeneMapper: Red, Green, Blue, Yellow, Orange. GeneMapper must register these objects in JNDI.
Panel	Combo box or browse box. List contains the registered panels from a GeneMapper installation. GeneMapper must register these objects in JNDI.
Size Standard	Combo box. List contains the size standard files associated with the GeneMapper installation. GeneMapper must register these objects in JNDI.
Run Protocol	Dakar-specific, combo box.
Analysis Protocol	Combo box. List contains the Analysis Protocols associated with the GeneMapper installation. GeneMapper must register these objects in JNDI.

AutoAnalysis Box Element	Description
Automatically analyze... checkbox	Checkbox. When true, sets Dakar to automatically analyze the samples.
GeneMapper 3.0 Installation combo box	Combo box. Sets the name of the GeneMapper 3.0 installation where Dakar should send the samples. List contains the JNDI registered GeneMapper installations.
Username field	Editable, alphanumeric. Contains the user name for the GeneMapper 3.0 installation.
Password field	Editable, alphanumeric. Contains the password for the GeneMapper 3.0 installation.

8.3 **GeneMapper Analysis Manager**

Each GeneMapper 3.0 installation shall have an application called the GeneMapper Analysis Manager. The purpose of this manager is to

- Listen for Dakar messages that indicate when samples are ready for analysis.
- Transfer the samples from Dakar to the queue and organize them based on Project Name and Username.
- Automatically launch GeneMapper 3.0 (when it is not in use), create or open the appropriate project, add the samples to the project, and start analysis.
- View details associated with the analysis
- Resubmit batches
- Print the analyzed sample data

GeneMapper 3.0 Analysis Manager				
File Help				
Project	Username	Samples	Date/Time In	Status
Project_001	HonebePC	240	Nov 15 2001 12:33 PM	Complete - Nov 16 2001
Project_002	HonebePC	240	Nov 15 2001 12:33 PM	Analyzing
Project_003	HonebePC	240	Nov 15 2001 12:33 PM	Waiting for GeneMapper 3.0
Project_004	YungKC	240	Nov 15 2001 12:33 PM	Error - Password Not Valid
<div> Details Resubmit Clear Completed Batches <input type="checkbox"/> Print Analyzed Samples </div>				

Element	Description
File menu	Contains items for: <ul style="list-style-type: none"> • Print... • Page Setup... • ----- • Exit
Help menu	Contains items for: <ul style="list-style-type: none"> • About GeneMapper 3.0 Automatic Analysis Queue

<u>Project Table</u>	<p><u>Lists the Project/Username combination that has pending samples for analysis.</u></p> <p><u>Functions:</u></p> <ul style="list-style-type: none"> • <u>Single click a row to select it.</u> • <u>Double click a row to view the Details dialog box.</u> <p><u>Columns:</u></p> <ul style="list-style-type: none"> • <u>Project = Name of the project. Note: There may be multiple projects of the same name listed if samples associated with those projects have different associated usernames.</u> • <u>Username = Username associated with the samples for a particular project. This was set in the Dakar plate document.</u> • <u>Samples = The count of the number of samples waiting the queue for the project.</u> • <u>Date/Time In = Date/time stamp when the batch was added to the queue.</u> • <u>Status = Status of the batch. Messages include:</u> <ul style="list-style-type: none"> • <u>Complete (date/Time)</u> • <u>Analyzing</u> • <u>Waiting for GeneMapper 3.0</u> • <u>Error – Password not valid, username not valid, etc.</u>
<u>Details button</u>	<u>Enabled when a batch is selected. Displays the Details dialog box.</u>
<u>Resubmit button</u>	<u>Enabled when a batch is selected. Displays the Resubmit dialog box.</u>
<u>Clear Completed Batches</u>	<u>Clears any session whose status is Complete. Alert: <i>Clear completed batches from the table? [Yes] [No]</i></u>
<u>Print Analyzed Samples checkbox</u>	<u>When true, sets the software to automatically print the sample files after they are analyzed.</u>

- 8 4 **Details Dialog Box** The Details dialog box displays the statistics associated with a batch as well as a log of processing events for each sample.

Details for "ProjectName"

Project: ProjectName
Username: HonebePC
Change...

Sampler: 240
Date/Time In: Nov 15, 2001 12:33
Status: Complete Nov 16, 2001 1:13 PM

Sample Name	Instrument ID	Run ID	Date/Time Analyzed	Result
Sample_001	Dakar_001	1234567890	Nov 15, 2001 12:33 PM	Analysis Successful
Sample_002	Dakar_001	1234567890	Nov 15, 2001 12:33 PM	Analysis Successful
Sample_003	Dakar_001	1234567890	Nov 15, 2001 12:33 PM	Analysis Successful
Sample_004	Dakar_001	1234567890	Nov 15, 2001 12:33 PM	Analysis Successful

Resubmit Close

Element	Description
<u>Project Detail Information</u>	<u>Read-only fields displaying information for:</u> <ul style="list-style-type: none">• <u>Project</u>• <u>Username</u>• <u>Samples</u>• <u>Date/Time In</u>• <u>Status</u>
<u>Change... button</u>	<u>Opens the Change dialog box. Enables the user to change the username/password for GeneMapper 3.0 access.</u> <div><p>Change Username/Password</p><p>Username: <input type="text"/></p><p>Password: <input type="password"/></p><p>OK Cancel</p></div>

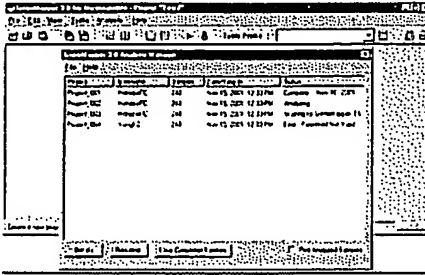
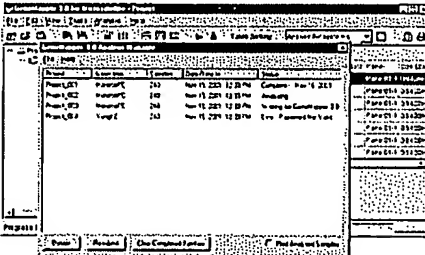
<u>Event Log Table</u>	<u>Displays the list of samples and their status.</u> <u>Columns:</u> <ul style="list-style-type: none"> • <u>Sample = Sample Name</u> • <u>Instrument ID = ID of the instrument where the sample ran.</u> • <u>Run ID = ID of the run for the sample</u> • <u>Date/Time Analyzed = Date/time stamp when the sample was analyzed.</u> • <u>Result = Status of the sample, such as Not Analyzed, Analyzed, Error – (Reason)</u>
<u>Resubmit button</u>	<u>Resubmits the batch to GeneMapper. Enable only if the batch's status is not Complete.</u>
<u>Close button</u>	<u>Closes the dialog box.</u>

8.5 GM Analysis Manager Interaction With GeneMapper 3.0

The GeneMapper Analysis Manager can only perform automatic analysis when the GeneMapper 3.0 application does not have a logged-on user. This means the GeneMapper Analysis Manager can take control of the GeneMapper 3.0 application when it is closed or when the Login to GeneMapper dialog box is visible.

For each batch in the GeneMapper Analysis Manager's project table, the following procedure is followed to analyze the samples.

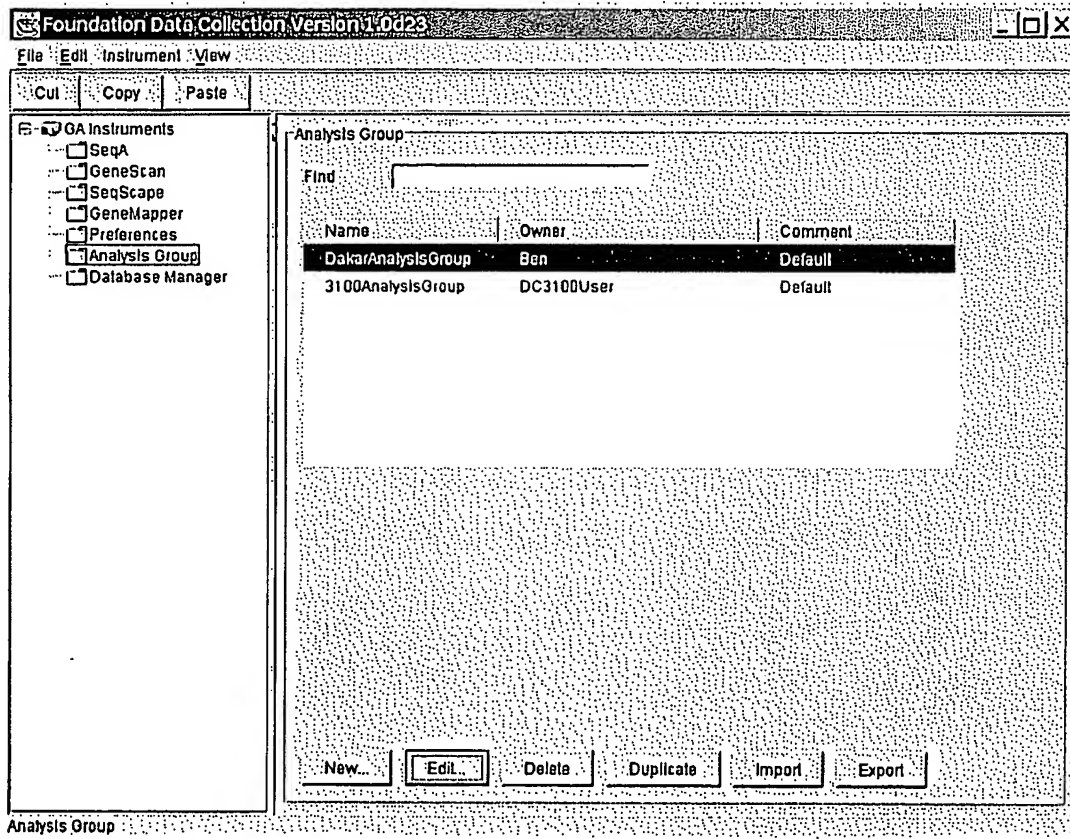
<u>Step</u>	<u>Action</u>	<u>Result</u>
1	<u>GM Analysis Manager selects the next batch in the project table.</u>	
2	<u>If closed, GM Analysis Manager launches GM 3.0.</u>	<u>GM 3.0 Login dialog box is shown behind the GM Analysis window.</u> <u>Note: If GM 3.0 is open and a user is logged-in, then the GM Analysis Manager cannot analyze the batch and must wait until GM 3.0 is closed or in the logout state. To inform the user, we should display a message in the Status bar: <i>GM Analysis Manager has pending batches to analyze. Please log out when convenient.</i></u>

3	<p>GM Analysis Manager attempts to log in using the username and password associated with the pending batch.</p>	<p>If successful, the GM 3.0 SDI window is shown behind the GM Analysis Manager window, then continues with Step 4.</p>  <p>If unsuccessful, GM Analysis Manager logs the error for the current batch, then returns to Step 1.</p>
4	<p>GM Analysis Manager determines if the batch's project exists.</p>	<p>If true, it opens the project in GM 3.0. If false, it creates a new project in GM 3.0, automatically naming it with the Dakar Analysis Group name.</p>
5	<p>GM Analysis Manager directs GM 3.0 to add the samples associated with the batch into the project</p>	<p>In GM 3.0, we see samples added to the Samples table. Note: GM 3.0 is locked and is not accessible by the user.</p> 
6	<p>GM Analysis Manager directs GM 3.0 to analyze the samples in the project.</p>	<p>In GM 3.0, we see the samples being analyzed. Note: GM 3.0 is locked and is not accessible by the user.</p>
7	<p>When analysis is complete, GM Analysis Manager directs GM 3.0 to save the project then log out.</p>	<p>The GM 3.0 window closes and the GM 3.0 Login dialog box is displayed. GM Analysis Manager then loops back to Step 1.</p>

Analysis Group Design Review



The Analysis Group is a concept that extends the idea of a project. It pulls together into a one set of settings all the information that Data Collection uses for extracting and analyzing samples after a run, and for passing the extracted samples to the downstream analysis applications.



1. Screen shot of Analysis Group Manager screen. From here we can create, edit, delete, duplicate, import and export analysis groups.

The screenshot shows a dialog box titled "Analysis Group Editor" with a close button (X) in the top right corner. The dialog has five tabs: "General", "Analysis", "Destination", "Notification", and "Naming". The "General" tab is selected. Inside the dialog, there is a large text area at the top labeled "Analysis Group Name". Below this, there are three text input fields: "Analysis Group Name" containing "DefaultAnalysisGroup", "Analysis Group Owner" containing "SomewhereInBarbados", and "Analysis Group Comment" containing "No Comment". At the bottom left, there is a checkbox labeled "Analysis Group Entry Completed" which is checked. At the bottom right, there are "OK" and "Cancel" buttons.

Field	Value
Analysis Group Name	DefaultAnalysisGroup
Analysis Group Owner	SomewhereInBarbados
Analysis Group Comment	No Comment

☒ Analysis Group Entry Completed

OK Cancel

2. Screen shot of the General pane of the analysis group editor. The checkbox labeled "Analysis Group Entry Completed" is checked by the user after the last plate containing members of this analysis group is edited. After a run, if this checkbox is on, and if a search for any remaining samples in this analysis group that have not been run comes up empty, then a notification that this analysis group is finished will be sent to the JMS topic.

Analysis Group Editor

General | **Analysis** | Destination | Notification | Naming

Analysis

☒ Do Autoanalysis

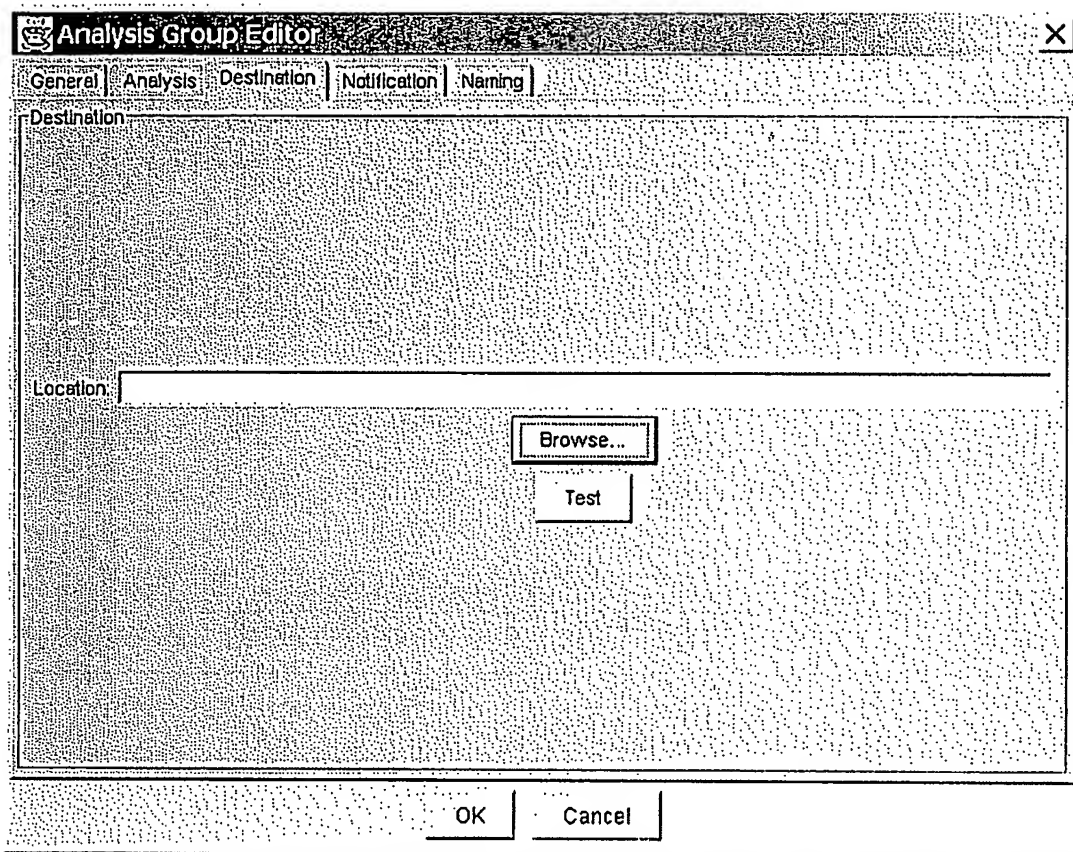
Analysis Type: SequencingAnalysis

Default Analysis Protocols for Runs 1 - 5

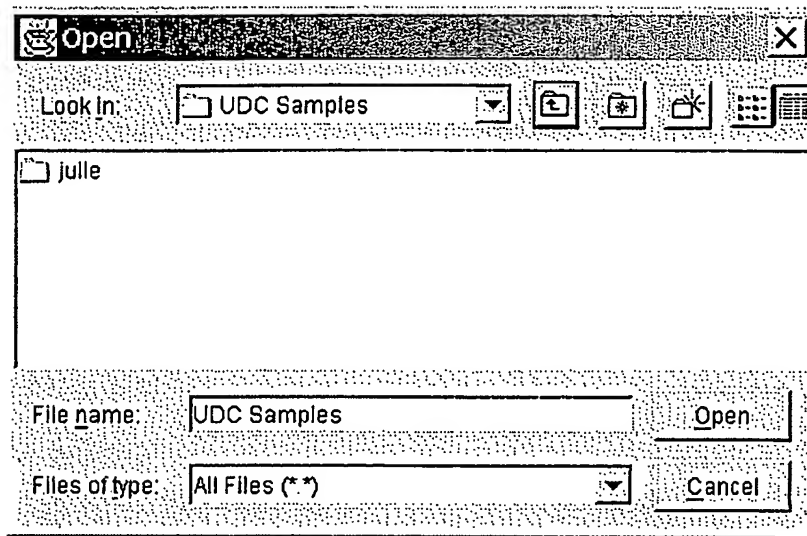
Run 1	BC-3100_SeqOffFlOff saz
Run 2	BC-3100_SeqOffFlOff saz
Run 3	BC-3100_SeqOffFlOff saz
Run 4	BC-3100_SeqOffFlOff saz
Run 5	BC-3100_SeqOffFlOff saz

OK Cancel

3. Screen shot of the Analysis pane of the analysis group editor. This allows the user to set the autoanalysis flag, the type of analysis, and the Default Analysis Protocols used for the 5 possible runs that a single plate can be used for. The Analysis Type and Analysis Protocol names come from the JNDI server. The version of the analysis Protocol is whatever is the latest one at the time that the plate is edited. These analysis protocols are only used for filling in the plate editor with these default values. The user can then override the entry by choosing a different one in the plate editor. These analysis protocols are not read from the analysis group at extraction time.



4. Screen shot of the pane for choosing a destination for the samples and folders created during extraction. The browse button allows a user to navigate to a folder on the local machine where the samples and folders are to be created. They are only created in that place, not in the run folder. A destination on a remote server can be chosen by mapping a remote folder to a drive letter, or by typing in the name of a server which is found in the local network. The test button attempts to create a file in the destination, and then delete it. The results of the test are displayed in a dialog.



5. When you hit the browse button, a standard file dialog is displayed.

Analysis Group Editor

General | Analysis | Destination | Notification | Naming

Notification

JMS Topic for Notification:

☐ Notify Sample Complete

☒ Notify Run Complete

☒ Notify Group Complete

Login ID jonesbb

Password @@@@@@

OK Cancel

6. The Notification pane allows the user to specify the types of notification that are desired for alerting the downstream apps that samples have been created. If the login ID and password fields have any contents, then they are included in the notification message. The message types are cumulative. All types that are checked are sent, even if a more inclusive message is sent.
 - a. If the Sample checkbox is checked, a notification is sent to the notification topic for each sample that is created.
 - b. If the Run checkbox is checked, a notification is sent to the notification topic after each run is complete for every analysis group that is used in the run. Each of these notifications contains a list of the samples belonging to that run and analysis group.
 - c. If the Group checkbox is checked, a notification is sent when the last sample in the group is complete. It contains a list of the samples in the analysis group.

Analysis Group Editor

General | Analysis | Destination | Notification | **Naming**

Sample File Name Format

Example: FilePrefix1.FileSuffix2.<None>

Prefix: FilePrefix1

Name Delimiter:

Format: <none>

Suffix: FileSuffix2

File Extension: <None>

Run Folder Name Format

Example: FolderPrefix3

Prefix: FolderPrefix3

Name Delimiter:

Format: <none>

OK Cancel

7. The Naming pane of the analysis group editor allows the user to specify the names of the samples and folders that will be used for the results of the run.
 - a. The File prefix is always prepended to the file name and the suffix is always appended.
 - b. The File Extension is determined by the type of application chosen in the "Analysis" pane.
 - c. As the user chooses elements for the name from the popup menus, additional popups appear for choosing additional elements, as seen in the next screen shot.
 - d. In the folder naming options, it is possible to choose "folder delimiter". This does not actually insert a character into the name. It instead creates a new subfolder using the elements of the name after that option.

Analysis Group Editor [X]

General | Analysis | Destination | Notification | Naming

Sample File Name Format

Example: FilePrefix1_A34_Mr.Holmes_Run_DummyInstrumentName_2000-07-31_6_Sample3_FileSuffix2.ab1

Prefix: FilePrefix1

Name Delimiter: _

Format:

Well Position	Owner Name	Run Name	Sample Name	<none>
---------------	------------	----------	-------------	--------

Suffix: FileSuffix2

File Extension: .ab1

Run Folder Name Format

Example: FolderPrefix3_ThePhiladelphiaProject\3_2002-04-21_05-45-15

Prefix: FolderPrefix3

Name Delimiter: _

Format:

Analysis G ..	Subfolder	Run Number	Date	Time	<none>
---------------	-----------	------------	------	------	--------

OK Cancel

8. As elements of the name are chosen additional popups appear to permit the choice of additional elements.

SeqScape 2.0 Integration with Foundation Data Collection Automatic Secondary Analysis

Preparing for SS Auto-Analysis

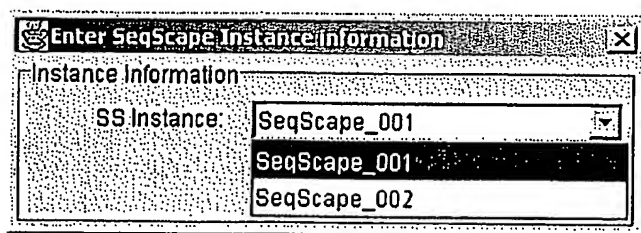
SeqScape (SS) organizes sample data in a hierarchical structure. A SeqScape Project can contain multiple Specimens, uniquely named within the Project. Each Specimen can contain any number of Samples, uniquely named within the Specimen. Data Collection must be able to provide this hierarchical information for each Sample in a run.

The following data are needed by SS to perform auto-analysis:

1. DC Run name
2. Username/password for the SS instance
3. List of sample files to analyze and their location(s)
4. Each sample file needs to contain the SS Project and Specimen it belongs to and any sample Analysis Protocols

The Data Collection user provides this information in the Plate Editor dialog. There is a different Plate Editor dialog for each application type (SeqA, SeqScape, GeneMapper, etc.), and there is a plate per instance of an application. It is necessary to select the SS instance first as the rest of the data necessary for the run comes from data the SS instance has placed in JNDI.

Note: We have discussed two possible ways to select the SS Instance. We could have one application instance per plate and have the user select the SS instance before showing the plate editor dialog. The other way is to have a SS instance column in the plate editor dialog. The user would have to first select the SS instance before being allowed to select the Project.



After the SS instance is selected the user is prompted to enter the other Sample information and auto-analysis data in the Plate Editor dialog. Much of the data in the plate editor table is derived from data the SS instance has placed in JNDI (described in the next section).

SS Plate Editor						
Well	Sample	Comment	Project	Specimen	AnalysisGroup	AnalysisProtocol
A1	Sample_000		Project_001	Specimen_001	AG_001	AP_001
B1	Sample_001		Project_001	Specimen_001	AG_001	AP_001
C1	Sample_002		Project_02	Specimen_001	AG_001	AP_001
D1	Sample_003		Create New Project	Specimen_001	AG_001	AP_001
E1	Sample_004			Specimen_001	AG_001	AP_001

The SeqScape Plate Editor contains special editor plug-ins for entering the Project, Specimen, and Analysis Protocol data.

The project column is editable via a cell editor plug-in. The cell allows the user to select an existing Project [from JNDI] for the SS instance or to create a new Project. If the user selects to create a new Project [in JNDI], the user is prompted to enter a Project name and select a Project Template [from JNDI]. The editor will also enforce a unique Project Name within the SS instance.

The Specimen column is also editable via a plug-in. The editor allows the user to either select from the list of Specimens [from JNDI] already in the Project, or create a new Specimen [in JNDI] in the Project. The uniqueness of the new Specimen name within the Project is enforced by checking it against the list of Specimens [from JNDI] already in the Project.

Each SS Project contains a Project Template, which contains all of the analysis settings, and defaults, including the default sample analysis protocol (called Analysis Default in SS). The user can override the Sample analysis settings on a per Sample basis via the Analysis Protocol editor in the Data Collection interface. The user must also be able to choose to use the default Sample analysis settings in the Project by not selecting an Analysis Protocol. The final plug-in is in the Analysis Protocol column. This plug-in allows the user to:

1. Select from the list of Analysis Protocols for the instance of SeqScape in JNDI
2. Select AND edit an Analysis Protocol from the list of Analysis Protocols for the instance in JNDI
3. Use the Project's default Analysis Protocol by NOT picking an Analysis protocol for the Sample
4. Create a new Analysis Protocol

This plug-in is the same as the one for SeqA.

In summary, the Project column in the Plate Record indicates the SeqScape Project and is selected from JNDI or newly created (by specifying a Project name and a Project Template), existing Specimens are selected from JNDI or new ones are created in the Specimen column, and the Sample analysis settings built into the SS Project will be used unless the user selects an overriding Analysis Protocol for the Sample in the Analysis Protocols column

An Analysis Group is assumed to contain the username and password to use with the application instance, locations of the sample files, and whether or not to auto analyze them.



Analysis Protocols shared across applications

The desire is the following: Files exported from SeqScape should be analyzable in SeqA using the same analysis protocol as was used by SeqScape.

It's thought that this shouldn't be an issue from an object point-of-view because the SeqScape 2.0 analysis protocol is currently a superset of the SeqA 5.0 analysis protocol. The whole trick will be in how we code this information in the sample file.

There's only one analysis protocol tag in the sample file. Its contents are an XML blob. Without introducing other analysis protocol tags (which we wouldn't like to do), there seem to be two solutions:

1. Have the SeqScape analysis protocol contain a SeqA analysis protocol. We would have to somehow guarantee the XML parser which turns the AP blob into the AP object could handle this kind of setup. (this code is owned by Ben Jones).
2. Store two analysis protocols concatenated in XML in the analysis protocol tag. One protocol would contain SeqScape specific tags and the other protocol would be identical to the SeqA 5.0 protocol. This would work as long as the XML parser used to create the analysis protocol object for SeqA can look for the right analysis protocol.



JNDI Data Structure

Much of the data that is used to fill in the columns in the Plate Editor comes from data in the JNDI for the instance of SS. The SeqScape data is stored in JNDI in the root/AnalysisApp context. Instances of SeqScape do not share any global data. The following is a description of how the SS data is structured in JNDI. The text in blue indicates a fixed name based on the agreement between FDC and analysis application. The text in black is populated at run time and the content presented below is for example purposes.

SeqScape [contains a list of SS instances]

Global&&& [part of JNDI structure – empty because SS has no global data]

SS_Instance1 [SS instance entry]

Projects [contains a list of Project names in the instance]

Project1

Specimens [contains list of Specimen names in the enclosing Project]

Specimen1

Specimen2

New Specimens [contains a list of new Specimen names created in DC]

Specimen3

Project2

Specimens

New Specimens

New Projects [contains a list of new Projects created in DC]

Project3

Project Template

PT1

New Specimens

S1

S2

Project Templates

PT1
PT2
Analysis Protocols
AP1
AP2
New/Modified Analysis Protocols
AP3

The SeqScope folder contains a list of existing SS instances

Each SS instance contains a list of Projects, New Projects, Project Templates, Analysis Protocols, and New/Modified Analysis Protocols.

Projects contain a list of Specimens and a list of New Specimens. New Specimens are those that are newly created in DC that have not yet been created in the official DD DataStore via auto-analysis.

New Projects are Projects that are newly created in DC that have not yet been created in the official DD DataStore via auto-analysis. In order to create a New Project, a Project Template must be specified (see description of a Project Template in the Project Templates description below). New Projects also contain a list of New Specimens.

Project Templates contains a list of the Project Templates available in the DataStore of the SS instance. SS Projects contain Project Templates, which contain all of the analysis and other settings for a Project.

Analysis Protocols are a list of Analysis Protocols (analysis defaults) in the DataStore of the SS instance.

New/Modified Analysis Protocols contains a list of Analysis Protocols newly created in DC that have not yet been created in the official DD DataStore via auto-analysis. Analysis Protocols that are edited in DC are moved from the Analysis Protocols list into the New/Modified Analysis Protocols list so the SS instance will know to update them in its master DataStore.

Messaging and Automation

Each installation of SS will have an associated daemon process that will always run in the background. The daemon registers itself in the JNDI with a JMS topic unique to the SS instance. The daemon listens for Run Completed messages from DC for the instance's JMS topic and manages the batch/automation processing.

The message from DC will contain the username and password to log into SS and the names and locations of all sample files to auto-analyze. It is assumed that FDC does not

require any progress updates or other acknowledgements other than that the message was received.

After receiving the message from DC, the daemon copies the sample files to the SS staging area (see Batch Processing Directory Structure below).

The daemon starts an instance of SS in headless mode using the username and password from the message and passes it the location of the Run data in the staging area.

SS opens the Project, loads the data from the staging area, and adds the Samples to the appropriate Specimens within the Project, creating new Specimens when appropriate.

The SS Analysis pipeline is run.

The project is saved.

Post processing (TBD) is performed. Possible post-processing activities include: Export or print reports, export alignments (NT or AA), export the project, export the log file

What if we receive the next message while one is still being processed? The daemon queues the request until the previous SS is finished (returns), then calls it again pointing it to the new data.

The daemon will have a GUI which can be shown to display the runs, their status, restart etc.

Possible auto-analysis error cases and resolutions:

Assuming that the DC plug-ins check for unique Project and Specimen names, there are only a few possible reasons why a sample file will not be correctly handled in batch processing and imported into a Project:

The Project has been deleted.

Resolution: Manually create the Project and either process the samples manually or re-run the batch

Attempt to create a New Project, but a Project with that name already exists. This is a problem because the user might have wanted different default settings than the existing Project.


Resolution: Rename the existing Project and re-run the batch or manually import the samples into the existing Project

The Sample name conflicts with an existing Sample in the Project/Specimen

Resolution: rename the Sample or manually import it to another Project/Specimen


Authentication errors

Resolution: create the user/password and re-run the batch or log into the GUI SS as an existing/valid user and process the sample data manually



SS synchronization with JNDI

It is important that the data in the SS instance's DataStore remains in sync with that in JNDI. It is also important that Project and Specimens be registered in JNDI as they are created in the Plate Editor so that the information can be used to populate the Sample information for following Samples. In order to simplify this synchronization, objects created in the DC Plate Editor are registered in special areas in JNDI - New Projects context, New Specimens (Project and New Projects sub-context), and the New/Modified Analysis Protocols context. When the SS instance creates a new object during auto-analysis, it is registered in JNDI and the associated New Project, New Specimen, or New/Modified Analysis Protocol is removed from JNDI.



Internal SS design

DataStore locking

Prior to 2.0, only one instance of SS could be run at a time. Locking the entire DataStore at start-up enforced this. In SS 2.0, the DataStore locking will be moved the Project level, allowing multiple instances to execute simultaneously, but the same project can not be open in multiple instances at any time.

There are two choices in terms of changing the DataStore locking. Just locking the Project objects is not enough, because multiple SS GUI instances can change top-level DataStore objects via the SeqScape Manager.

One option is to lock each top level DataStore object (Project, Project Template, RDG, Analysis Default, Display Setting).

The other option is to still lock the DataStore as a whole for SS GUI instances (enforcing only one GUI can run at a time) and also locking Project objects. The headless SS will not check for the DataStore lock, but will check for Project locks. Whenever a project is opened, it is locked and the lock file contains the text indicating justification for locking it. For example "This Project is locked for batch processing of run X at Y time/date" or "This Project is locked by SeqScape by user X at Y date/time". If the headless version has the Project locked and the GUI version tries to open or edit the Project, a dialog is displayed showing the lock file justification string and the user has the option of

overriding the lock. This must be an option because lock files can be left behind if SS or the machine crashes or loses power, etc.

Secondary Analysis

Running multiple instances of SS will be expensive memory –wise as the entire DataStore is currently read in at start-up. The headless/batch version could only load the necessary parts of the DataStore. Another solution, if SS were already running, would be to have the daemon send messages to the running SS to perform secondary analysis in the background.

SeqScape maintains a circular log file for reporting warnings and errors during execution. Multiple instances would require multiple log files. We could enforce that only one headless instance can run at a time (though this is not really necessary as we can just name the log files uniquely), and that the headless version would keep its log file separately from the GUI version.

Batch Processing Directory Structure

Each installation of SS will have a folder for queuing batch inputs and results:

<SS installation Directory>/data/automation

Top-level batch processing folder

<SS installation Directory>/data/automation/tobeprocessed

Folder of subfolders containing sample files, one per run to be processed

<SS installation Directory>/data/automation/results

Folder of subfolders, one per batch processed run, containing the results from batch processing. Results may include: Log file, exported reports or alignments (NT/AA), sample files that could not be processed due to name collisions, project already open, etc.

Internal Processing

Daemon

Daemon receives a RunComplete message from DC on the JMS Topic associated with the instance of SS

Daemon creates a new folder for the run in the SS instance's tobeprocessed folder and copies into it the sample files from the run.

[REDACTED]

The daemon starts the SS instance in headless (automation) mode

SS headless(automation) mode

On Initialization, headless SS does not load the whole DataStore, just the list of existing Projects

[REDACTED]

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☒ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.